神经网络架构搜索的阅读总体概述

从 2019 年的 DARTS 开始，NAS 进入了 one-shot 的时代，之前的基于强化学习（用控制器的方式）的 NAS 和基于进化学习的 NAS，由于成本消耗巨大，逐渐淘汰，近些年来的 NAS 论文大多都是基于梯度更新的方式。

DARTS 以连续松弛的思想实现了权重共享，从来实现了梯度更新，大大减少了训练成本。但是 DARTS 存在着各种各样的问题，比如在搜索阶段需要非常大的显存，比如 DARTS 在搜索结束后（已确定网络架构）需要重头开始训练，比如 DARTS 在不同数据集上会出现不稳定，倾向于选择 skip-connect 的问题等等，这些问题的后续讨论，产生了一系列新的改进。比如 2020 年 PC-DARTS 的减少信道的措施，2022 年 β-DARTS 提出了新的正则方式……

与此同时，基于 one-shot 出现了一种新的思想，不妨一次训练所有网络，即 Two-stage，（NAS 可从训练+搜索的角度去考虑），使用权重共享/蒸馏等方式训练一个很大的超网（由于搜索空间的不同，这个超网可以有几千个子模型），然后将这个超网训练的很好，直接提取超网中的子网进行部署。在训练阶段结合准确率预测器和 flops（或者其他限制）预测器再确定最优的模型。比如 2020 年 BigNAS 提出的三明治法则，比如 2021 年 Attentive NAS 采样训练支配上下界的子网络……

在阅读最近的论文时，还发现 NAS 的搜索目标不仅仅是基于 Accuracy，由于考虑到部署到移动设备上，NAS 常常还需要考虑延迟，flops 等多目标，将其加入损失函数，由此也有部分论文比如 FBnet 等等，当然这部分通常放入实验部分，不太算是 NAS 的一种搜索思想。

关于神经网络架构搜索的改进方向(一些愚见)

1.可以在 DARTS 改进方向和 two-stage 思路上两个角度做进一步优化，是否能够提出更好的搜索策略，来实现计算量和准确率的优化，这部分读到的论文很多。
2.关于搜索空间，考虑更大更复杂的搜索空间，卷积核的长，宽，步长，参数，层数，信道数……会怎么样，或者能不能实现将卷积，循环融合到一个搜索空间中去，能不能将 transformer 加进去，这部分读到的论文很少。
3.神经结构搜索算法在更多任务中评估表现，因为 NAS 的很多论文都是在 imagenet 和 cifar 数据集上做的，都是图像类的，不同的数据集例如文本？是否会有不同的表现。这部分读到的论文也很少
4.架构参数和模型参数的不平衡，如何更好地更新参数问题。
5.在现有的 NAS 方法上进行通用的加速研究，NAS 搜索需要的时间和计算资源过大。

（下面是关于最近读的 NAS 方向论文的阅读报告）

DARTS：2019 ICLR 会议论文

阅读原因：

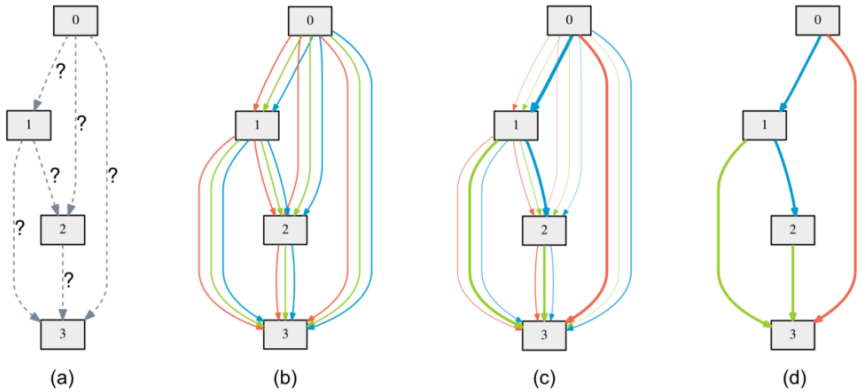
这篇文章是 one-shot 神经网络架构搜索的开山之作，之前在神经网络架构搜索中，由于搜索空间本身是离散型的特点，所以常常是离散化的搜索（基于强化学习或者进化学习），这篇文章是最早提出连续化搜索的方法之一，尤其是他的开源代码更是对后续的研究起到了很大的促进作用。本人实习期间是基于 DARTS 代码中的卷积算子做了一些工作，所以再重新对其第二次阅读。

论文内容概述分析：

从摘要中可以看出（our method is based on the **continuous relaxation** of the architecture representation, allowing efficient search of the architecture using gradient descent.）DARTS 的核心就是连续松弛，这个操作开上帝视角看等价于一个 softmax（）的操作，而在神经网络中 softmax 是可以通过梯度更新的方式进行反向传播的，所以通过这个角度他就将离散化的搜索变成了连续型的搜索。

详细来看 DARTS 的搜索空间：DARTS 神经网络中的每一层就是一个 cell 算子。为了对照卷积神经网络和循环神经网络。DARTS 有两类算子，分别是卷积算子和循环算子。每一个卷积算子有一个输出，有两个输入，每个输入是前两个 cell 的输出；每一个循环算子有一个输入和一个输出。由此，搜索整个神经网络架构就转变成搜索成 ceLL 算子即可。图 d 即为最终的一个算子。

每个算子有两部分构成分别是节点和操作。节点可以理解为卷积神经网络中的特征图，即图 a 的 0，1，2，3。操作即卷积操作，池化，激活函数等等，每两个节点之间有多个候选操作，这就是搜索空间。连续松弛就是在每个候选操作上加一个架构超参数，这个架构超参数经过 softmax 之后即表示选中这个操作的概率。然后更新参数的方式为两级最优化，在训练集上更新模型本身的参数，在测试集上更新架构参数，最终使其收敛。



论文优点以及改进方向：

1. 提出了基于连续松弛的梯度更新的方式，提出了可微的神经网络结构搜索。且可应用于卷积和循环的神经网路结构。

2. 在减少 gpu 训练消耗的前提下，在 cifar10 上得到了很好的精度。

3. 可迁移到其他的数据集，例如 ImageNet 并且获得较好的结果。

PC-DARTS：2020ICLR 会议论文

阅读原因：

基于 DARTS 的改进版本，通过这篇论文想了解一下 DARTS 的某些不足之处。

DARTS 的问题：

DARTS 需要搜索完模型本身的参数和架构参数后再重新训练。所以它是用 cifar 搜索出来的神经网络结构，然后再去 imagenet 数据集上训练。即他只能在小数据集上搜索，如果增大 batch_size 后，会导致 GPU 不够。

论文内容概述分析

为了解决 DARTS 在搜索过程中显存占用过大和计算量的问题，这篇文中提出了一种思路，即用超网的一部分来进行搜索，具体是每个候选操作的输入都是部分信道。例如每次都是所随机选取有信道的 1/4 输入卷积，然后最后的结果再与刚开始剩余的 3/4 相加。即下面公式

$$f_{i,j}^{\mathrm{PC}}(\mathbf{x}_i;\mathbf{S}_{i,j})=\sum_{o\in\mathcal{O}}\frac{\exp\left\{\alpha_{i,j}^{o}\right\}}{\sum_{o'\in\mathcal{O}}\exp\left\{\alpha_{i,j}^{o'}\right\}}\cdot o(\mathbf{S}_{i,j}*\mathbf{x}_i)+(1-\mathbf{S}_{i,j})*\mathbf{x}_i. \tag{1}$$

但是这样搜索出来的结果会变得不稳定，因为每个特征图的很大一部分来源于之前的特征图，所以模型会更倾向于选择 pool，skip 而不是卷积。（This regularizes the preference of a weight-free operation (e.g., skip-connect, max-pooling, etc.) over a weight-equipped one (e.g., various kinds of convolution) in O. In the early stage, the search algorithm often prefers weight-free operations, because they do not have weights to train and thus produce more consistent outputs, i.e., o(xi). In contrast, the weight-equipped ones, before their weights are well optimized, would propagate inconsistent information across iterations.），这个会造成准确率的大幅度降低，所以他引入了 early normalization 的概念，给每个节点的输入来源，即每个比他小的节点乘以权重参数，来保证可以使得经过每个候选操作后特征图差别较大的目的。由于减少了通道，所以可以增加 batch_size 进行训练，最终可以直接搜索 imagenet 的网络结构。

论文优点以及改进方向：

1.基于 DARTS 的论文，对 darts 搜索过程中显存不足的问题的解决提供了一种思路。

2.在 imagenet 上搜索训练，可以在较短的时间内用较少的资源得到较高的结果。

3.减少通道和 dropout 与调小 batch_size 的差别不是特别大，都是让数据的一部分不参与训练。

BigNAS：2020ECCV 会议论文

阅读原因：

读 Attentivenas（2021CVPR）的过程中发现有很多概念不太清楚，这些概念来自于 Big NAS，所以对其进行查阅整理，发现其相比于上面连续松弛与池化这个概念，提供了一个新的神经网络结构搜索的概念。即同时训练一个含有很多个子模型（可以达到几千个模型）的大的超网。每个子模型通过蒸馏，参数共享的方式来进行更新。

论文内容概述分析：

很多神经网络架构搜索的模型往往需要在搜索完成后进行重新训练、微调或其他后处理，但这些步骤会增加计算的复杂度，增加计算成本。BigNAS 的提出，让搜索出的模型在不经过额外的 fine-tune 或者其他后处理操作，也能取得 SOTA 的效果。

BigNAS 的大模型包含很多个小模型，通过改变(kernel and channel sizes, network depths, input resolutions)来决定模型，这些小模型在搜索完之后就可以都达到不错的准确率。那么问题就在于如何得到一个比较好的超网。为此 BigNAS 提出了一系列的技巧，也可以看做是搜索策略。分别是三明治法则，蒸馏,初始化、学习率调整和正则化(network sampling during training, inplace distillation, network initialization, convergence behavior and regularization）

具体做法是：在每个训练阶段，采样 1 个最大的子网，1 个最小的子网，再随机采样两种大小的网络。在更新整个模型的权重之前，再把所有采样模型的梯度聚合在一起。这样可以提高最小模型的性能下界和最大模型的性能上界。然后输入数据喂入最大的模型，得到预测标签，再将这个标签喂入小模型。由于搜索空间中的所有子网都是残差网络，BigNAS 将每个残差块的最后一个 BN 层的 γ 参数设置为 0，让每个残差块的 output 初始化为全 0 张量。BigNAS 将学习率设置跌到一开始的 5%就不再变化了。除此之外，正则化只在最大的那个采集出的子模型上实施。

然后文章还有一部分工作，因为他是一个硬件资源感知的 nas 方法，所以他会考虑再 latency，flops，存储量的约束下选择模型，那么 BigNAS 在搜索完超网之后，需要根据约束条件选择最佳的网络结构。作者提出了粒度从粗到细的采样策略：首先我们找到最有希望的候选网络结构的大概模样，然后在每个感兴趣的框架体系结构附近用较细的粒度采样不同的网络结构。注意的是这一步不需要重新训练，只需要选择超网的权重即可。

论文优点以及改进方向：

1.给了一种全新的视角来选择神经网络架构，即没有连续松弛，而是通过蒸馏，权重共享的方式训练一个包含很多个子模型的超网。

2.确实解决了之前 NAS 重新训练的问题，只需要单独搜索即可。而且提出的三明治法则，大模型小模型怎么同时训练，正则化等很多细节策略有很多启发意义。

3.搜索空间较为丰富。

AttentiveNAS：2021CVPR 会议论文

阅读原因：

是神经网络结构搜索 two-stage 方向的论文之一，对训练和搜索阶段做了更进一步的优化，来实现更好的帕累托边界。

论文内容概述分析：

整理总结了两阶段搜索。第一阶段先在超网中训练，得到一个准确率很高的超网（超网中的每个子网准确率都很高），第二阶段就是搜索（选择），满足在特定限制例如 flops 要求下的的网络模型。

AttentiveNAS 在 BigNAS 上改进了抽样策略，BigNAS 是采样一个最大和最小的模型加入到损失函数中的正则。而 Attentive 选择采样一批子模型，采用 BestUp + WorstUp 策略，BestUp 用于改进当前最好的帕累托前沿，WorstUp 用于提升当前表现最差的候选网络。分别提高上限和下限。

然后就是提出了两种方法来有效采样最佳或最差的帕累托前沿。为了（减少计算开销）。一是是使用 mini-batch 损失的性能评估器，对于每个网络结构，在当前 mini-batch 训练数据上测量的训练损失，用它作为性能度量。二是预训练精度预测器作为性能评估评估器，在验证集上训练准确性预测器，然后，对于每个网络结构，使用精度预测器给出的预测精度作为性能估计。

考虑到在之前的两阶段神经网络架构搜索中，训练和搜索阶段的联系并不是很紧密，在搜索阶段会考虑到各种硬件需求，则将这些要求加入到训练过程中（类似于之前的 FBnet，Proxylseenas 那种加上延迟 latency 正则）

论文优点以及改进方向：

1.提出 attentiveNAS，使用不同的采样策略 BestUp + WorstUp，改进了最优和最差的帕累托前沿。（其实还是在两阶段的这个大思路下，然后对搜索策略做了一些尝试）。

2.提出了两种方法来有效采样最佳或最差的帕累托前沿。

3.在 FLOPs 约束条件下，成为在 ImageNet 上表现最好的 NAS 模型。

β-DARTS：2022CVPR 会议论文

阅读原因：

随机开盲盒，阅读 2022 年顶会论文。

论文内容概述分析：

β-DARTS 是基于 2019 年最初的 DARTS 模型进行了算法上的优化。最初的 darts 存在两个问题，一个是搜索的候选操作容易倾向于出现 skip-connected 候选模块，另一个是在某个数据集上搜索出来的架构不一定适应不同的数据集。（这篇文章采用的还是最初的 DARTS 模型，其实后面出现了很多的方向，包括不用再次重新训练等等。）

β-DARTS 主要工作就是在损失函数中加入了一种新的正则，这个正则跟常规的 L2 正则化不同，他是用架构 a 参数经过 softmax（）之后得到的参数 b，然后将其进行变化，作为新的正则。放在代码中其实非常少。如下图所示

Algorithm 1 PyTorch Implementation in DARTS

1: $\mathcal{L}_{Beta}$ = torch.mean(torch.logsumexp(
self.model._arch_parameters, dim=-1))

2: loss = self._val_loss(self.model, input_valid,
target_valid)+$\lambda\mathcal{L}_{Beta}$

然后这篇文章用了大量的公式推导来证明这样做的意义（这边不详细展开），然后证明在一些数据集上确实得到了不错的结果。

论文优点以及改进方向：

1. 基于最初版 DARTS 的方向仍然有继续在做的论文。